Porting Of Applications From Silverlight To HTML5

Aksel Kornesjö
2016

Bachelor of Science in Engineering Technology
Computer Game Programming

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering
Porting Of Applications From Silverlight To HTML5

Aksel Kornesjö

Department of Computer Science, Electrical and Space Engineering

June 2016
Abstract

This report describes a project about porting a Silverlight based resource management and scheduler application called Adoxa into HTML5 and JavaScript. It includes redesigning and implementing the front end interface and making it work with the existing underlying backend. Frontend frameworks were researched, and a AngularJS application implemented. A few problems encountered and solutions to them are described. A new frontend was created successfully, and is presented at the end of the report.

Sammanfattning

Preface

This was the largest project I have ever worked on, with challenges and new concepts to
learn. Though I had some prior experience with AngularJS that was used in this project,
I still learned a lot about it. During this project I also familiarized myself with many
different web frameworks, and good practices in web development. Overall it has been very
educational and interesting. I want to thank Jens for helping me with the project and
guiding me through their existing platform.
Contents

1 Introduction 1
  1.1 Background ........................................ 1
  1.2 Goals and Purposes .............................. 2
  1.3 Limitations .................................... 2

2 Materials and Methods 3
  2.1 Frameworks ..................................... 3
    2.1.1 AngularJS .................................. 3
    2.1.2 LumX ....................................... 3
    2.1.3 Bootstrap .................................. 4
    2.1.4 FullCalendar .............................. 4
    2.1.5 KendoUI ................................... 4
  2.2 Front end Structure .......................... 4
    2.2.1 Angular Applications ........................
      2.2.1.1 Controllers .............................
      2.2.1.2 Directives ..............................
  2.3 Back end Communication ....................... 6
    2.3.1 Fiddler ....................................
    2.3.2 C# HTTP API ................................
  2.4 Configuration ................................ 7
  2.5 Visual Design ................................ 8
    2.5.1 CSS/LESS ..................................
      2.5.1.1 Sourcemaps ..............................
    2.5.2 W3 Standards ................................
    2.5.3 Mobile Support/Responsive design ........
  2.6 Project Structure ............................... 9
    2.6.1 Bundling ...................................
      2.6.1.1 "use strict" ............................
    2.6.2 File Structure .............................
  2.7 Validation ..................................... 13

3 Result 14
  3.1 Application Layout ............................ 14
  3.2 Scheduler Section ............................. 14
    3.2.1 Booking Dialog and Other Pages .............
4 Discussion and Conclusion 18
  4.1 Code Structure ................................................................. 18
  4.2 UI Discussion ................................................................. 18
    4.2.1 Booking Dialog UI ....................................................... 19
    4.2.2 Administration UI ...................................................... 19
  4.3 Future Work ................................................................. 19
  4.4 Social and Ethical Considerations ....................................... 20
  4.5 Conclusion ................................................................. 20
List of Figures

3.1 Navigation Menu ................................................................. 14
3.2 Scheduler View ................................................................. 15
3.3 Default Booking Dialog with One Section Open ...................... 16
3.4 Administration Page ........................................................... 17
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSS</td>
<td>Cascading Style Sheet</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>CSHTML</td>
<td>C# Razor HTML</td>
</tr>
<tr>
<td>NPAPI</td>
<td>Netscape Plugin Application Programming Interface</td>
</tr>
<tr>
<td>JS</td>
<td>JavaScript</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>AJAX</td>
<td>Asynchronous Javascript And XML</td>
</tr>
<tr>
<td>MVW</td>
<td>Model View Whatever</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
</tbody>
</table>
1 Introduction

Internet based applications, also known as Rich Internet Applications, have recently become popular[1]. JavaScript is mainly used for these applications, but before that was possible other frameworks and plugins had to be used. A brief history of relevant information about the internet follows.

1.1 Background

During the early days of the internet, web pages were used to convey static information through static HTML pages with images, and later gained some interactivity with the use of Javascript[2].

NPAPI was the next step towards a richer web experience, with which developers could create applications pulling the web and desktop applications closer together by being able to run native code inside the browser. But with the increased freedom and merging of native applications on the web, the risk for security vulnerabilities grew as well. This was especially true for NPAPI since it only worked as a thin layer between the browser and native system[3].

Silverlight is one plugin built upon NPAPI. It was released in 2007 with a handful of features including displaying media such as pictures and videos. Functionality was eventually expanded to a full featured web framework similar to the likes of Adobe Flash[4]. However, the popularity of Silverlight never reached the same levels of Flash[5]. It was used primary due to its powerful video streaming capabilities by large companies such as Netflix [6]. In the advent of HTML5 and improved media capabilities brought with it, the role Silverlight played was phased out. Due to this, and the deprecation of NPAPI by modern browsers[7], Microsoft officially ended support for Silverlight[8]. Their advice for developers was to use other solutions than Silverlight.
1.2 Goals and Purposes

The goal of this project was to design and implement a new frontend for an already existing web based application. The old frontend is soon to be obsolete, since it relies on Silverlight. The backend also needed to be connected with the new frontend. The application to be ported, Adoxa[9], is a resource management program developed by Explizit[10]. It is used to display and manage reservations and bookings for different resources, such as facilities, equipment and employees. The application can show a calendar overview for resources, so users can see when a specific resource is available or occupied. Registered users in Adoxa are assigned to teams. Bookings belonging to a team a user is not a member of can not be accessed. A resource can be a member of multiple teams.

As the UI should be consistent and intuitive, some research in UI design was made. For this Google Material[11] was an inspiration, along with a couple of UI frameworks that were tested and compared.

1.3 Limitations

Since the application will be web based, the main limitations come with that. Compared to native applications, web based programs have some limitations[12]. However, with the advent of HTML5 and modern web technologies such as WebGL[13] and JavaScript Web Workers[14], the boundaries between desktop applications and web based counterparts are closing in on functionality.
2 Materials and Methods

2.1 Frameworks

During the project, some research into different web frameworks was made. Some of the primary frameworks that were examined are listed below. Not all of these frameworks were used, but were used as inspiration when designing the application.

2.1.1 AngularJS

A major part of the application is AngularJS, which is a MVW framework. The framework is developed by Google. It is currently one of the most popular MVW JavaScript frameworks[15], with a large community. The primary purpose of the framework is to separate website user interfaces from their underlying logic and functionality, and provide a solid architecture for single page applications. AngularJS also offers a powerful two way data binding system, meaning that a variable can be bound to an HTML element, and upon change in the element it will also be updated in the JavaScript code and vice versa.

2.1.2 LumX

The plan was to make use of the Google Material Design specifications, so it would be best to find a front-end framework for AngularJS that worked well across web browsers, especially Internet Explorer. The first framework that was tested was Angular Material[16]. It worked very well initially, and seemed good since it is an official AngularJS project. It was easy to use and looked very promising. However performance was bad in Internet Explorer, with some modules in it not working at all.

An alternative framework named LumX[17] was found after some research. It is similar to Angular Material, and is under active development. Although it not being very widely used, it seemed mature enough to use in this application, so it was used for a while.
Since the main layout was done using custom stylesheets, the changing of UI frameworks was relatively easy. Other frameworks were also tested and used as inspiration, but not used in the final version. These included Twitter’s Bootstrap, and FullCalendar.

### 2.1.3 Bootstrap

Bootstrap is a responsive user interface framework developed by Twitter[18]. It is the most popular JavaScript and HTML UI framework, with the second most stars on Github at the time writing this report[19]. Initially the application was written using the framework. It was later switched out for a custom solution during the use of Google Material Design. This was due to the Google Material frameworks weren’t compatible with Bootstrap.

### 2.1.4 FullCalendar

The calendar view was central to the whole project, so choosing a mature framework for this was important. When researching this, the first choice would have been FullCalendar[20]. However, some features, including viewing two calendars showing two separate resources’ bookings next to each other, was not possible out of the box. This was a feature that some of the client wanted, so a more suitable library would be ideal.

### 2.1.5 KendoUI

Later on in the project a switch from FullCalendar to KendoUI[21] was made. This was due to FullCalendar not having all necessary features available. Kendo also features a rich set of user interface components and widgets, such as date and time pickers, and multiselects. This was used in the final version of Adoxa. The primary downside of KendoUI is that it is not free-to-use, but has to be purchased. A trial of it was tested, after which a license was purchased by Explizit. It was worth it, due to the well documented codebase and guides in how to use the framework, and the simplicity of incorporating it into the existing code.

### 2.2 Front end Structure

AngularJS already provides a basic structure for separating the view from the application logic. The application consists of a number of controllers, services, and directives. What these are will be briefly explained below.
2.2.1 Angular Applications

AngularJS is a full front-end framework for creating single page applications. At the most basic level, only one empty Angular module is needed to make the application run. A single page application is an application that loads the page traditionally at first, after which additional content is loaded and displayed with the use of JavaScript. This speeds up the response times of the application, since only certain parts are loaded at a time, instead of reloading the whole page for each view change.

\[ \text{Listing 2.1: AngularJS Application attached to HTML} \]

In listing 2.1 an application is initiated using an module named \textit{ModuleName}. This alone however does not add any functionality; for that Angular Controllers and Directives are used.

### 2.2.1.1 Controllers

Controllers are instantiable objects, which are bound to HTML elements in the application view. Upon binding the controller to the element, data can be assigned to the Controller scope, which can then be displayed in the view.

\[ \text{Listing 2.2: JavaScript Controller} \]

The example in listings 2.2 and 2.3 show the text defined in the Controller named TestController. The content in double curly brackets are parsed as AngularJS expressions; in this case just a variable is printed.
2.2.1.2 Directives

Directives are intended to be used for the modification of the HTML element itself instead of controlling the data of a view. For the data, a Controller can be assigned to the Directive, which will share the scope. With directives it is also possible to create custom HTML element tags, which could make HTML code written simpler to read.

2.3 Back end Communication

The original Adoxa backend ran in a separate server, which then had to be communicated with. This was accomplished using HTTP requests to C# WepAPI Controllers that ran with the ASP.net site.

Since a Silverlight frontend already existed, it was possible to examine the service calls it made to the service, and then simply implement them as a C# WebAPI[22] controller.

2.3.1 Fiddler

To easily see what was happening behind the curtains of the Silverlight application, a program called Fiddler[23] was used. Fiddler enabled easy monitoring and analysis of HTTP calls. This simplified the rewriting of the Silverlight application by using the Silverlight client, and observing which service calls were made in Fiddler. After this an HTTP WebAPI Controller could be made and then replicate the service call within it.

2.3.2 C# HTTP API

As an backend service already existed, a means of communicating with it had to be developed. The exchange is based on HTTP requests, and for this project a mostly REST[24]-based scheme was implemented. REST stands for representational state transfer. Basically it means that resources are accessed by their specific URI, and are then interactable with HTTP calls. This made the API easy to test and understand. For example, to get info about all facilities, a GET request could be made to /API/Facility. The service returns a JSON object containing an array with all facilities. Getting a specific facility by the facility ID would be achieved by a GET request to /API/Facility/FacilityID. This is how most of the API worked, with some exceptions such as the booking search.

In this case a GET request would be made, with additional URL encoded parameters, as shown in listing 2.4.
GET /API/BookingSearch?From=2016-01-01&To=2016-01-02

LISTING 2.4: HTML

When the API is written in this way, it is simple to test API calls directly in the browser; just type in the API address, and the call result will be shown in the browser window.

When updating or saving objects, a POST request is sent, as seen listing 2.5.

POST /API/Facility [Request body contains Facility object]

LISTING 2.5: HTML

And lastly, to delete an object, a DELETE request is sent (listing 2.6).

DELETE /API/Facility/5

LISTING 2.6: HTML

Using already existing requests defined in the HTTP protocol for the API is recommended; there is no reason for reinventing the wheel.

2.4 Configuration

Since multiple customers with different needs use the software, a means of configuring features in it was implemented. The Silverlight version already had XML based configuration files depending on where it was deployed. During runtime, the settings flags were loaded from the existing configuration file before the application started, after which they were stored in a global constant in Angular. The configuration flag object was also exposed in the root scope, which made it easy to check for flags within application views, as seen in listing 2.7.

<div ng-if="ApplicationFlags.IsAdministrationEnabled"></div>

LISTING 2.7: HTML with Application Flag

The above element will only be visible if the administration menu flag had been set to true in the service backend. This separation of configuration makes it much simpler to manage which features will be available for customers, because no changes have to be made in the frontend code to hide and show different features; it is all done server side. Reusing the configuration system made it possible to use the same config files as before when deploying the application to customers, with no need for creating new configurations and complications.
2.5 Visual Design

It was essential for the application to look and feel modern and easy to use. The process of designing the application was very iterative, meaning that during the project the visual appearance and layout changed multiple times. For inspiration, the Google Material design specification was read.

2.5.1 CSS/LESS

While HTML specifies the markup of a web page, it does not specify its visual appearance. For that, Cascading Style Sheets (CSS) are used[25]. CSS in itself is quite rigid and can be quite cumbersome to work with. One main reason for this is that it does not use variables. LESS[26] however solves this. It is a language that compiles into CSS. There are a couple of ways to do this, either by precompiling LESS files into CSS and then including the compiled file in a web page, or by using a JavaScript library to compile LESS files during page load. In this project the former option was chosen with the use of Gulp.

Gulp is a streaming build system, in which a set of tasks are defined in a JavaScript file. It can be set to watch files, and then process files through a pipeline. Gulp pipelines can process the provided data freely in any way by using Gulp plugins. What the plugins do is very varied, from simply copying files to starting a server that hosts a website for debugging. Currently there are over 1800 plugins available[27], and new ones are being submitted regularly. For this project, only a simple LESS file watcher/compiler was needed. A simple Gulp pipeline that used the plugins gulp-watch and gulp-less-sourcemap was written. With this, all LESS files in a specified folder were monitored for changes, and then compiled into their corresponding CSS files upon modification.

2.5.1.1 Sourcemaps

Since multiple LESS files are compiled into a single CSS file, finding where a style is defined when inspecting elements in the browser can be tricky. This is also true for minified and concatenated JavaScript files. It can be solved by generating sourcemap files during LESS file compilation. The sourcemap files can then be loaded alongside the compiled file. Sourcemap define where every part of the compiled code comes from, which the web browser developer tools then simply use to look up where the code comes from[28].
2.5.2 W3 Standards

Since most websites should be accessible by everyone, a set of standards have been defined by the World Wide Web Consortium for developers and designers to follow\cite{29}. These were taken in mind when designing the application. Since the project makes use of third party libraries such as Kendo UI, it can be difficult to know whether or not they fulfill these standards.

2.5.3 Mobile Support/Responsive design

Originally the website was only supposed to work on desktops, but since mobile website applications have risen in popularity, as well as a request from customers, the application was made mobile friendly\cite{30}. This was solved by creating different layouts, one for large screens, and one for smaller ones.

2.6 Project Structure

Initially, the project and file structure was simple. Angular controllers, directives, services and views were put into separate directories. With only a few controllers and views this was manageable, but once the complexity and size of the application grew, it became increasingly convoluted to maintain the project and find specific files. Adding JavaScript source files also needed to be included in the main HTML file in a script tag. To solve this problem of structural complexity, the bundling capabilities in ASP.net\cite{31} were of great help.

2.6.1 Bundling

First off, bundles were defined by giving them a name and files to be included. A basic JavaScript bundle could be defined as shown in listing 2.8.

```csharp
bundles.Add(new ScriptBundle("~/bundles/js")
.Include("Scripts/*.js");
```

**Listing 2.8: Example Bundle Creation**

This snippet would create a bundle named `/bundles/js` and contain all the JavaScript files in the Scripts directory. The bundle could then be included in a CSHTML file (listing 2.9).
Now once the webpage is requested, ASP.net will generate script tags automatically for every file in the JavaScript bundle. Depending on build mode, minification and concatenation of the files will also occur. This will result in that only one script tag will be generated for the bundle, improving load times and decreasing network bandwidth. There are a few things to take into consideration when writing JavaScript files that will then be concatenated and minified. The first thing is that since all the files will be placed in one single large one, variable names and code scopes could easily clash, causing unwanted behaviours or even crashing the application. If for example, the following two files would be concatenated together (listing 2.10 - 2.11).

File 1:

```javascript
function myFunction() {
    console.log("A");
}
$(element).click(myFunction);
```

Listing 2.10: JavaScript File 1

```javascript
Function myFunction()
{
    console.log("B");
}
```

Listing 2.11: JavaScript File 2

The expected behaviour would be that when the element is clicked, the console output would be A. This is however not what will happen, since JavaScript parses functions first before doing anything else, so the second myFunction would overwrite the first one before the click listener is added. Using immediately invoked function expressions is one simple way to solve this problem. This entails wrapping all code inside an anonymous function which is then immediately called. It effectively creates a separate enclosed scope for every JavaScript file, as in listing 2.12.

```javascript
(function(){
    function myFunction() {
```
console.log("A");
}
$(element).click(myFunction);
})();

Listing 2.12: Wrapping Code in Immediately Invoked Functions

Now the code inside the anonymous function will have its own scope, and there should be no worries about getting variables mixed up as long as they are declared within unique function scopes.

Another issue is variable scrambling, which is when the minifier changes variable names to much shorter counterparts to make output file smaller. This is generally not an issue because variable names are mostly there for code readability. But sometimes the name of a variable needs to be known even by the code. Listing 2.13 shows an example.

```javascript
angular.module("controllers")
 .controller("TestController", TestController);
function TestController($scope) {
  $scope.message = "Hello";
}
```

Listing 2.13: AngularJS Controller

A JavaScript code minifier could change the $scope parameter to something shorter, making it impossible for AngularJS to inject the $scope component into the controller. The are a few ways to solve this, but in this project the $inject variable was used (listing 2.14).

```javascript
...
TestController.$inject = ["$scope"];
function TestController($scope) {...}
```

Listing 2.14: The $inject Variable

AngularJS will always look for an $inject variable in functions during runtime to reliably figure out which services should be injected into it. These were the main issues ran into when minifying and concatenating the JavaScript source files.

2.6.1.1 ”use strict”

It is recommended to add the ”use strict”[32] declaration at the top of JavaScript files as in listing 2.15.
"use strict";
var i = 0;
p = 4;

Listing 2.15: The use strict definition

The "use strict" definition is parsed by newer JavaScript interpreters and adds restrictions in how variables can be defined and modified. In the above example the variable "i" is correct, but "p" will result in an error, since all variables have to be declared with the var keyword before being used. The restrictions added by the define are good for keeping the code readable, and making error checking more robust.

2.6.2 File Structure

The file structure evolved during the project along with the complexity of the application. Initially file names had CamelCased names, with JavaScript files that could contain more than one component each. It quickly lead to confusion when trying to find a specific Angular controller or service. After discovering and reading through the Angular style guide[33], some changes to the code were made. Significant changes included limiting the amount of code in JavaScript files, and having a unified file name format. For example, a Angular controller named HomeController would be placed in a file called home.controller.js. It would only contain this and nothing more. This naming system plays very well with the JavaScript bundling too, as the file names follow an easily organized pattern. All different sections were placed in separate directories. They contain JavaScript relevant to the page with names such as *.controller.js, *.config.js, *.service.js and so on. They could all then be imported in correct order with the rule in listing 2.16.

```c
bundles.Add(new ScriptBundle("~/bundles/js")
    .IncludeDirectory("~/Content", "*.config.js", true)
    .IncludeDirectory("~/Content", "*.controller.js", true)
    .IncludeDirectory("~/Content", "*.directive.js", true));
```

Listing 2.16: JavaScript Bundling

This will recursively include all files in the Content directory, matching the file patterns specified. It made it effortless to add new modules to the application; simply create a new directory, and populate it with the module scripts and views.
2.7 Validation

When building a web application with a lot of server communication there are a plethora of things to take into consideration. One of these considerations is whether the client or server should handle most of the validation. Logically, the service should handle all validation of data sent to it, and respond accordingly. This is how the existing backend works, which simplifies the client side validation significantly. If the client sends invalid data, an error object is returned which the client then uses to display a suitable error message. However, some client side validation is good to have to improve the responsiveness of the web application. Validations such as required input fields, time and date field restrictions and correct email formatting are easily implemented using standard HTML5 input attributes[34] and AngularJS validation[35]. Here below a sample form is presented, displaying how basic validation can be achieved using AngularJs. See listing 2.17 - 2.18.

```html
<form name="sampleForm" ng-submit="submitForm(sampleForm)">
  <input type="email" id="emailField" required="true" ng-model="sampleEmail" />
  <div ng-if="sampleForm.emailField.$invalid">
    Email field is incorrect
  </div>
</form>

Listing 2.17: HTML Form with AngularJS Directives

```javascript
$scope.submitForm = function(form) {
  if(form.$invalid) {
    console.log(form.emailField.$error);
  }else{
    console.log("Form is valid.");
  }
}

Listing 2.18: AngularJS Controller Function for Form
```

The form has a single input field with the type set to email. The form will be invalid until the email field has been correctly filled in. Using these validations along with server-side resulted in a stable validation system, which showed the end user detailed and easily understandable error messages.
3 Result

3.1 Application Layout

The application consists of a number of pages with subpages. When the application starts, a user profile and application settings are loaded, which contains a list of flags of which features of the application should be available. This will hide or show specific parts of the application, to match the customer’s requirements.

Navigation is done in the header (fig. 3.1). Subpages are also displayed in this view.

![Figure 3.1: Navigation Menu](image)

3.2 Scheduler Section

The most work was put into this view, since it is the central and most used section in the application. It consists of the main scheduler view, in which bookings can be filtered by resource. In this view it is also possible to edit and create bookings, either by drag and dropping or manually editing the booking using the booking editor.

There are three different types of bookings, all visible in figure 3.2. A normal booking, reserved time, and a published booking. On the left side, the resource selection menu is seen. Only bookings for resources that are selected are shown in the scheduler view. On the left side of bookings, a vertical bar with letters is visible, indicating which resources are participants of the booking. The full name of the resource is shown upon hovering over the letters.
3.2.1 Booking Dialog and Other Pages

The booking dialog (fig. 3.3) differs depending on what data the client needs. A basic booking only contains the booking start and end time, whether or not it is recurring, an activity and which employees are attached to it. Some clients need more info attached to bookings. These sections are toggled in the configuration file in the back-end.

The calendar administrators have access to an administration page (fig. 3.4), where most of Adoxa can be configured. New resources and teams can be created in this view, along with other administrative tools are available in it.

These are a few pages in Adoxa, displaying the general visual look and layout of the application.
**FIGURE 3.3: Booking Dialog**

![Booking Dialog](image)

<table>
<thead>
<tr>
<th>Aktivitetstyp</th>
<th>Beskrivning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test (15 min)</td>
<td></td>
</tr>
</tbody>
</table>

Tillåter parallella bokningar
(Permitted parallel bookings)

Återkommande bokning
(Recurrence)

Starttid: 13:00
Sluttid: 13:15
Datum: 2016-05-05
Figure 3.4: Administration Page
4 Discussion and Conclusion

4.1 Code Structure

In the beginning the file names and the way the code was written was not very well structured, but instead focused on getting basic functionality to work. It was soon obvious that it would have to change to keep the code maintainable in any grade. Since JavaScript is so loosely typed it is very easy to get confused by what type a variable is and where it is. Some modern IDEs try to solve this by automatic predictions of variable names and functions, which makes it a little bit easier[36]. The proneness for errors is still relatively significant and a good way to lose time that could have been avoided. A very good resource discovered while reading online was the Angular Style Guide[33] which not only covers good practices when writing AngularJS applications, but also JavaScript in general. The style guide contains contributions from over a hundred developers and is endorsed by the Angular creators.

4.2 UI Discussion

The user interface structure was loosely based on the former silverlight version. Moving too far from it could lead to confusion in customers, and the feel of the application. The application has been tested by a few existing customers, and changes were made based upon their critiques and opinions. The resulting UI was well received by the clients.

The general design became more flat and non round shaped than Google Material design. In the end, the design ended up as more of a custom one, rather than following the guidelines of Google Material. This was a conscious decision since the clients wanted to be able to see a lot of information at a time. The spacious design of Google Material was not very suitable for that, thus resulting in the development of a compact UI, which was appreciated by the customers. As LumX was based on Google Material, it was more or less phased out in exchange for Kendo UI with a custom theme and own components.
An additional detail worth mentioning is that even if the LumX framework initially seemed mature enough to use in production, it did cause a few problems during development. These problems such as cross browser inconsistencies and JavaScript bugs and glitches slowed down development and the general stability of the application. To avoid these problems, it is recommended to use very popular, well tested frameworks such as Bootstrap.

4.2.1 Booking Dialog UI

The booking dialog UI consists of a list of sub-items of the booking as seen in figure 3. Subitems are loaded into the dialog depending on application flags and booking data. This was a decent solution for bookings with a lot of data, but ideally some other way to display bookings would be better, since it looks quite crowded and slightly confusing. One solution could be to use a tab format instead, which would show the sub-item titles at the top or bottom of the booking dialog.

4.2.2 Administration UI

At first, the administration menu had a secondary top navigation bar which contained all administration pages. It was later switched out for a navigation bar on the side as seen in the figure before. The change was due to the top bar becoming too cluttered when the amount of administration pages increased, sometimes even making the navigation bar spanning multiple rows. The sidebar solved that problem, and it also looked better and was easier to navigate. On smaller screens, the sidebar transforms back into a top bar, but with only the active menu visible. Upon hovering or clicking it, a navigation drop down appears, which contains the sub pages.

4.3 Future Work

As with most projects, there is room for improvement. The dynamic nature of JavaScript objects can get very confusing once the codebase grows larger. There already exists strongly typed languages that compile into JavaScript. One of these languages is TypeScript. To get it to work with already existing JavaScript code can be tricky though, since it needs definition files for the JavaScript to know what data types in the code are and other info that is not defined in the normal JavaScript. Definition files already exist for many popular JavaScript frameworks, such as jQuery and AngularJS. The successor of Angular, V2, even comes with TypeScript support out of the box. This would be a good start for future improvements.
4.4 Social and Ethical Considerations

The primary purpose of the application is to manage resources and data that might have private information. During development a test database was used with manufactured users and data, so that real data would not be compromised. The debug environment serves the application through HTTP, completely unencrypted. However, deployed client versions are secured with HTTPS, ensuring connections to be safe.

As the application works on mobile devices, clients can easily examine or make bookings on the go. When bookings are made, it is possible to print out a appointment verification to be sent to a client. Another available option is to send these verification through SMS or email, saving paper and time.

4.5 Conclusion

In conclusion, a front-end replacement for the initial Silverlight application was successfully implemented. The communication with the existing backend service was achieved with a REST-based API. The structure of the front-end application makes it easy to add new functionality to it. Most of the functionality is in working condition, and has been tested. Issues such as some peculiarities when handling recurring bookings have to be looked into, but it does have more to do with the back-end.
Bibliography

[1] Popularity of internet applications, 05 2016. URL


[5] Flash and silverlight usage comparison, 5 2016. URL
https://w3techs.com/technologies/details/cp-silverlight/all/all.


[9] Adoxa webpage, 5 2016. URL
http://www.explizit.se/produkter-tjanster/adoxa/.


https://www.google.com/design/spec/material-design/introduction.html.


[33] Angularjs style guide, 5 2016. URL

[34] Html5 input validation attributes, 5 2016. URL
   http://www.w3schools.com/tags/att_input_min.asp.

[35] Angularjs validation documentation, 5 2016. URL
   https://docs.angularjs.org/guide/forms.

[36] Intellisense and javascript, 5 2016. URL